

# A Time-Optimal Parallel Algorithm for 3D Convex Hulls \*

Nancy M. Amato<sup>†</sup>  
Coordinated Science Laboratory and  
Department of Computer Science  
University of Illinois  
1308 W. Main St.  
Urbana, IL 61801  
email: amato@cs.uiuc.edu

Franco P. Preparata<sup>‡</sup>  
Department of Computer Science  
Brown University  
Box 1910  
Providence, RI 02912  
email: franco@cs.brown.edu

## Abstract

In this paper we present an  $O(\frac{1}{\alpha} \log n)$  time parallel algorithm for computing the convex hull of  $n$  points in  $\mathbb{R}^3$ . This algorithm uses  $O(n^{1+\alpha})$  processors on a CREW PRAM, for any constant  $0 < \alpha \leq 1$ . So far, all adequately documented parallel algorithms proposed for this problem use time at least  $O(\log^2 n)$ . In addition, the algorithm presented here is the first parallel algorithm for the three-dimensional convex hull problem that is not based on the serial divide-and-conquer algorithm of Preparata and Hong, whose crucial operation is the merging of the convex hulls of two linearly separated point sets. The contributions of this paper are therefore (i) an  $O(\log n)$  time parallel algorithm for the three-dimensional convex hull problem, and (ii) a parallel algorithm for this problem that does not follow the traditional paradigm.

## 1 Introduction

Convex hulls are one of the most fundamental geometric constructs. In addition to being of considerable interest in their own right, convex hulls are often useful in solving apparently unrelated problems in Computational Geometry. Therefore, considerable research effort has focused on developing algorithms, both serial and parallel, for computing convex hulls.

The sequential complexity of computing the convex hull of a set  $S$  of  $n$  points in  $\mathbb{R}^d$ ,  $d \geq 2$ , is known to be  $\Omega(n \log n)$  (see, e.g., [PS85]). Although there exist several optimal serial algorithms for this problem when  $d = 2$  (refer, e.g., to [Gra72, And79]) and  $d = 3$  [PH77], optimal parallel algorithms are known only for  $d = 2$  [AG86, ACG<sup>+</sup>88, AG88, MS88, Che90]. In general, a parallel algorithm is said to be *optimal* if it runs in time  $O(\log^c n)$ , for some constant  $c$  (often referred to as polylogarithmic, or polylog time) and the product of the time and the number of processors used (the processor-time product) is of the same order as the sequential complexity of the problem.

---

\*This paper was presented in preliminary form at the 9th Annual ACM Symposium on Computational Geometry, San Diego, CA, May 1993.

<sup>†</sup>The work of this author was supported in part by an AT&T Bell Laboratories Graduate Fellowship, the Joint Services Electronics Program (U.S. Army, U.S. Navy, U.S. Air Force) under contract N00014-90-J-1270, and NSF Grant CCR-89-22008.

<sup>‡</sup>The work of this author was supported in part by NSF Grants CCR-91-96152, CCR-91-96176, and ONR Contract N00014-91-J-4052, ARPA order 8225.

Similarly, a parallel algorithm is said to be *efficient* if it runs in polylog time and the processor-time product exceeds the sequential complexity of the problem by at most a polylog factor.

When working in  $\mathfrak{R}^3$ , several parallel algorithms have been proposed: the first due to Chow [Cho80] required  $O(\log^3 n)$  time and  $O(n)$  processors, Aggarwal *et al.* [ACG<sup>+</sup>88] proposed a new algorithm with these same time and processor bounds, Dadoun and Kirkpatrick [DK89] implemented the algorithm of Aggarwal *et al.* more efficiently, and Amato and Preparata [AP92] gave an algorithm using  $O(\log^2 n)$  time and  $O(n)$  processors. It is only recently that an optimal  $O(\log^2 n)$  time parallel algorithm for the three-dimensional convex hull problem was obtained by Goodrich [Goo93]. To date, there is no known efficient deterministic parallel algorithm for computing the convex hull of a three-dimensional point set in  $O(\log n)$  time. However, Reif and Sen [RS92] give an optimal randomized algorithm that achieves  $O(\log n)$  running time with probability  $\geq 1 - n^{-c}$ , where  $c \geq 1$  is a constant; Goodrich's optimal  $O(\log^2 n)$  time deterministic parallel algorithm is obtained by using his new geometric partitioning technique to derandomize Reif and Sen's algorithm.

All of the above algorithms use the CREW PRAM model of parallel computation; the CREW PRAM is a shared memory model in which each shared memory location can be accessed in unit time and concurrent reads from (but not writes to) the same shared memory location are allowed (for details of the various PRAM models consult [KR90]). We note here that  $\Omega(\log n)$  time is required to compute the convex hull of a  $d$ -dimensional point set,  $d \geq 2$ , on an exclusive write PRAM, i.e., an EREW PRAM (no concurrent reads or writes) or a CREW PRAM. This follows from (i) the fact that computing the convex hull of a planar point set, even if the points are sorted by  $x$ -coordinate, is at least as hard as computing the "OR" function, and (ii) the work of Cook, Dwork, and Reischuk [CDR86] which established that computing the "OR" of  $n$  bits on an ideal CREW PRAM requires  $\Omega(\log n)$  time regardless of the number of processors used.

Another class of convex hull algorithms worthy of notice are the *output-sensitive* algorithms whose running time is a function of  $h$ , the size of the output; note that while  $h = O(n)$  in both the two- and three-dimensional case, it may be as small as  $O(1)$ . Sequential output-sensitive convex hull algorithms are known. Kirkpatrick and Seidel [KS86] give an optimal  $O(n \log h)$  planar convex hull algorithm. Edelsbrunner and Shi [ES91] established that  $O(n \log^2 h)$  time is sufficient to find the convex hull in  $\mathfrak{R}^3$ . Subsequently, Chazelle and Matoušek [CM92] have given an algorithm for the three-dimensional case that achieves optimal  $O(n \log h)$  time; their algorithm is obtained by derandomizing the optimal  $O(n \log h)$  algorithm of Clarkson and Shor [CS89]. The algorithms of Kirkpatrick and Seidel and Edelsbrunner and Shi can be implemented in parallel on an ARBITRARY CRCW PRAM: the two-dimensional algorithm runs in  $O(\log^2 n)$  time and achieves a processor-time product of  $O(n \log h)$ , and the three-dimensional algorithm runs in  $O(\log^3 n)$  time with a processor-time product of  $O(n \log^2 h)$ . Goodrich and Ghouse [GG91] give randomized algorithms for the CRCW PRAM that solve the two-dimensional convex hull problem in  $O(\log n)$  time while performing  $O(n \log h)$  work, and the three-dimensional version in  $O(\log^2 n)$  time while performing  $O(\min\{n \log^2 h, n \log n\})$  work; for both algorithms the stated running time is achieved with probability  $\geq 1 - n^{-c}$ , where  $c \geq 1$  is a constant.

Thus, it remains an important open problem to find a deterministic algorithm for computing the convex hull of a three-dimensional point set in  $O(\log n)$  time using  $O(n)$  processors. Although the deterministic  $O(\frac{1}{\alpha} \log n)$  time and  $O(n^{1+\alpha})$  processor algorithm, for any constant  $0 < \alpha \leq 1$ , we present here is still sub-optimal, it has the important feature that it achieves optimal  $O(\log n)$  time on a CREW PRAM for the three-dimensional convex hull problem, whereas all prior deterministic

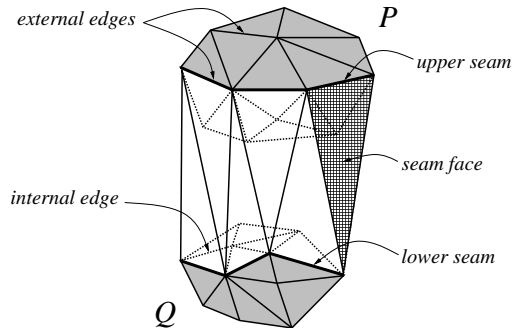


Figure 1: The merging phase of the sequential three-dimensional convex hull algorithm.

parallel algorithms for this problem use at least  $O(\log^2 n)$  time.<sup>1</sup>

All traditional parallel algorithms for the three-dimensional convex hull problem are based on the serial divide-and-conquer algorithm of Preparata and Hong [PH77]. (Recent exceptions are the randomized algorithm of Reif and Sen [RS92], and its derandomized version due to Goodrich [Goo93].) Let  $CH(S)$  denote the convex hull of the point set  $S$ . The serial algorithm for computing the convex hull of a point set  $S$  is outlined in Algorithm 1.1.

---

**Algorithm 1.1 Sequential-3D-Convex-Hull( $S$ )**

- Step 1.** Partition  $S$  into two sets,  $P$  and  $Q$ , such that the  $z$ -coordinate of each point in  $P$  is greater than the  $z$ -coordinate of every point in  $Q$ .
  - Step 2.** Recursively compute  $CH(P)$  and  $CH(Q)$ .
  - Step 3.** Compute the cycle of supporting faces that are tangent to  $CH(P)$  and  $CH(Q)$ , and merge  $CH(P)$  and  $CH(Q)$  along this cycle to form  $CH(S)$ . The cycle of supporting faces shares with each subhull a circuit of edges, called a *seam*. (See Fig. 1.)
- 

Any algorithm inspired by the above paradigm (referred to here as “bisect-and-conquer”, to stress the subdivision into *two* subproblems) necessarily consists of  $O(\log n)$  merge phases. Since it seems unlikely that two linearly separated convex hulls can be merged in constant time, any parallel algorithm based upon this approach seems doomed to require time  $\omega(\log n)$  (i.e., greater than logarithmic). Thus, for some time it has been suggested in the research community that only a finer subdivision than bisection is likely to improve the time performance; our algorithm is the first to achieve such an objective. In particular, the algorithm presented in this paper avoids the above drawback by departing from the bisect-and-conquer paradigm as follows: instead of dividing the point set  $S$  into two subsets, we partition it into  $O(n^\alpha)$  subsets, each of size  $O(n^{1-\alpha})$ , recursively construct the convex hull of each subset, and then merge the resulting  $O(n^\alpha)$  convex hulls to form  $CH(S)$ ,  $0 < \alpha \leq 1$ . This multi-partitioning scheme is a technique used in many

---

<sup>1</sup>Related independent and concomitant research has been reported in [PDW92], which outlines a (substantially different) technique claimed to achieve the same time and processor complexities as the one we present here.

parallel algorithms (see, e.g., [Jáj92, Rei93]), and is sometimes referred to as *many-way divide-and-conquer*; for example, the optimal parallel planar convex hull algorithms mentioned above [AG86, ACG<sup>+</sup>88, AG88, MS88] employ this technique.

Our approach greatly alters the overall strategy of the merging phase of the algorithm as follows. Note that in the traditional bisect-and-conquer approach, each face of the merged hull contains at least one edge from one of the subhulls; indeed, since the surface of the convex hull is assumed to be triangulated, a face of the merged hull is either a face of one of the subhulls, or it contains a seam edge of one of the subhulls and a seam vertex of the other subhull. Thus, the merging process in the bisect-and-conquer approach can be accomplished by classifying each edge (or face) of the subhulls as either external or internal to the merged hull. However, in our case, a face of the merged hull need not contain an edge of any of the subhulls, and thus the merging process must classify each *vertex* of one of the subhulls as either internal or external to the final merged hull, i.e., the traditional edge classification must be replaced by vertex classification.

Thus, the contributions of this paper to the understanding of the parallel three-dimensional convex hull problem are as follows. First, we depart from the bisect-and-conquer paradigm that has dominated previous parallel algorithms for this problem; a necessary consequence of our new approach is the development of a criterion for classifying vertices, rather than edges or faces, as either internal or external to the convex hull. Secondly, and perhaps more importantly, we have shown that it is indeed possible to compute the convex hull of a point set in  $\mathbb{R}^3$  in  $O(\log n)$  time with a relatively small number of processors.

We finally note here that there exists a trivial algorithm for computing the convex hull of a three-dimensional point set  $S$  in  $O(\log n)$  time using  $O(n^4)$  processors on a CREW PRAM, where  $|S| = n$ . This algorithm is face-based, rather than edge- or vertex-based. We determine the faces of  $CH(S)$  as follows. For each of the  $\binom{n}{3} = O(n^3)$  subsets  $S' \subset S$  of cardinality three, determine if they form a face of  $CH(S)$  by checking to see if all points of  $S$  lie on one side of the plane containing  $S'$ . This takes  $O(\log n)$  time using  $O(n)$  CREW PRAM processors for each subset  $S'$ . Then, the order of the faces around each vertex of  $CH(S)$  can be found in  $O(\log n)$  time using  $O(n)$  processors. Thus, the entire process takes time  $O(\log n)$  using  $O(n^4)$  processors on a CREW PRAM.

## 2 Breaking the $O(\log^2 n)$ Barrier

In this section we present a relatively simple algorithm that computes the convex hull of  $n$  points in  $O(\log n)$  time using  $O(n^2)$  processors on either an EREW or a CREW PRAM. Although this algorithm does much more work than the algorithm we shall present in the next section, it allows us to introduce some useful notation and illustrate techniques that are also used in the more efficient algorithm to be presented later. The following definition formalizes the concepts of internal and external.

**Definition 2.1** Let  $S$  denote a point set in  $\mathbb{R}^3$ . A point  $v \in S$  is said to be *external* if it is a vertex of  $CH(S)$ ; similarly, a point  $v \in S$  is said to be *internal* if it is not external, i.e., it is contained in the interior of  $CH(S)$  (or, equivalently, there are four points in  $S - \{v\}$  such that  $v$  is internal to their determined tetrahedron). If  $v$  is external to  $CH(S)$ , then (the ordered sequence of) the other external vertices that are incident to  $v$  on  $CH(S)$  will be referred to as  $v$ 's *neighborhood*, and will be denoted by  $n(v)$ ; the order of the sequence  $n(v) = (n_0, n_1, \dots, n_{k-1})$  is so assumed that the vertices  $v, n_i$ , and  $n_{(i+1) \bmod k}$  determine a face of  $CH(S)$ ,  $\forall 0 \leq i < k$ .

In order to compute the convex hull of a point set  $S$  in  $\mathfrak{R}^3$ ,  $|S| = n$ , we note that it is sufficient to classify each  $v \in S$  as either internal or external to  $CH(S)$ , and if  $v$  is external to compute the sequence  $n(v)$ . Once we have determined the convex hull vertices, and their neighborhoods, it is a simple matter to form the standard doubly-connected-edge-list (DCEL) [PS85] representation of the convex hull; recall that in a DCEL we record, for each edge, its two endpoints  $v_1$  and  $v_2$ , the two faces incident to it, and the edges which follow it in a clockwise traversal around the edges incident to  $v_1$  and  $v_2$  on  $CH(S)$ . Thus, in order to show that  $CH(S)$  can be computed in  $O(\log n)$  time using  $O(n^2)$  processors, it is sufficient to show that, using  $O(\log n)$  time and  $O(n)$  processors, each point  $v$  of  $S$  can be classified as either internal or external, and that, if  $v$  is external, then the sequence  $n(v)$  can be computed within the same time bound.

We now discuss how a vertex  $v \in S$  can be classified as either internal or external, and how the sequence  $n(v)$  can be found if  $v$  is external. Let  $v \in S$ , and define  $r(v, p)$  as the ray originating at  $v$  and passing by some other point  $p \in S$ . Next, define the set of rays  $R_v = \{r(v, p) \mid p \in S - \{v\}\}$ . The following simple lemma is the geometric basis of our algorithm.

**Lemma 2.1** *A vertex  $v \in S$  is external to  $CH(S)$  if and only if  $v$  is external to  $CH(R_v)$ . [In particular, if  $v$  is internal to  $CH(S)$ , then  $CH(R_v) = \mathfrak{R}^3$ , and if  $v$  is external to  $CH(S)$ , then  $CH(R_v) \neq \mathfrak{R}^3$  and  $v$ 's neighborhood  $n(v)$  lies on the boundary edges (rays) of  $CH(R_v)$ .]*

**Proof:** Note that  $CH(R_v)$  either has the single finite vertex  $v$  or it coincides with  $\mathfrak{R}^3$ . Assume that  $v \in S$  is internal to  $CH(S)$ . Then there are four points of  $S$ , say  $p_1, p_2, p_3$ , and  $p_4$ , whose tetrahedron contains  $v$  in its interior; it follows that the convex combination of the four rays  $\{r(v, p_i) \mid i = 1, 2, 3, 4\}$  coincides with  $\mathfrak{R}^3$ , i.e.,  $v$  is internal to  $CH(R_v)$ . Conversely, assume that  $v$  is external to  $CH(R_v)$ . Then, for any four other points of  $S$ , their determined tetrahedron does not contain  $v$ , thus proving that  $v$  is also external to  $CH(S)$ . ■

Thus, in order to classify  $v \in S$  as either internal or external to  $CH(S)$ , it is enough to find  $CH(R_v)$ , and moreover, if  $v$  is external, then  $CH(R_v)$  will yield the sequence  $n(v)$ . Although the problem of computing  $CH(R_v)$  may seem to be as difficult as computing  $CH(S)$ , as we will see below, the fact that all rays in  $R_v$  originate at  $v$  greatly simplifies matters. The above discussion yields Algorithm 2.1 for classifying any  $v \in S$  as either internal or external to  $CH(S)$ .

---

**Algorithm 2.1** CLASSIFY( $v, S$ )

**Step 1.** Construct  $R_v = \{r(v, p) \mid p \in S - \{v\}\}$ .

**Step 2.** Compute  $CH(R_v)$ .

**Step 3.** If ( $CH(R_v) = \mathfrak{R}^3$ )  
     **then** return (*internal*)  
     **else** return (*external* and  $n(v)$ )

---

The correctness of CLASSIFY follows directly from Lemma 2.1, and thus we need only determine its complexity. We let  $|S| = m$ . We first note that Steps 1 and 3 can be implemented in  $O(1)$  time using  $O(m)$  processors.

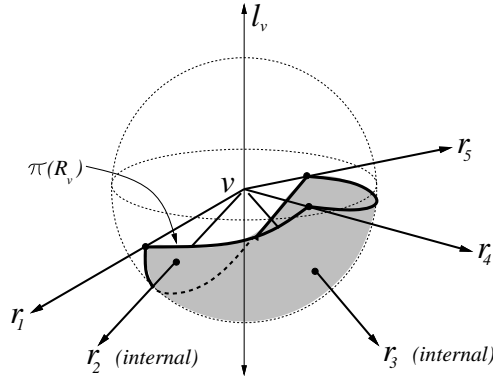


Figure 2: The spherical polygon  $\pi(R_v)$ .

If  $CH(R_v) \neq \mathfrak{R}^3$ , any plane  $H$  intersecting  $CH(R_v)$ , intersects it in a (possibly unbounded) convex polygon. In particular, consider a ray  $r \in R_v \cap CH(R_v)$ , and let  $u(r)$  be the point of  $r$  at unit distance from  $v$ . Then the plane  $H(r)$  orthogonal to  $r$  and containing  $u(r)$  intersects  $CH(R_v)$  in a convex polygon  $P(r)$ . The edges of  $P(r)$  incident on  $u(r)$  belong to lines tangent to the unit sphere  $\Sigma$  centered at  $v$ . It is realized that, when  $CH(R_v) \neq \mathfrak{R}^3$ , the intersection of  $CH(R_v)$  with  $\Sigma$  is a spherical polygon  $\pi(R_v)$ , whose edges are arcs of  $\Sigma$ 's great circles, so that their tangents at  $u(r)$  contain the edges incident upon  $u(r)$  in  $P(r)$  (see Fig. 2). The optimal  $O(\log m)$  time parallel planar convex hull algorithms [AG86, ACG<sup>+</sup>88, AG88, MS88] can be adapted to construct the spherical polygon  $\pi(R_v)$ . For the sake of completeness, we sketch the adapted version of the optimal CREW PRAM planar convex hull algorithm that was discovered independently by Atallah and Goodrich [AG86] and Aggarwal *et al.* [ACG<sup>+</sup>88]. Although the optimal EREW PRAM planar convex hull algorithm of Miller and Stout [MS88] can easily be adapted to the present scenario, we omit the details since it is slightly more complicated than the CREW algorithms [AG86, ACG<sup>+</sup>88] and because the algorithm described in the next section also requires a CREW PRAM version of CLASSIFY.

The computation of  $\pi(R_v)$ , and thus of  $CH(R_v)$ , is now described. We devise a system of polar angles with origin at  $v$  as follows. (We assume here that  $|R_v| \geq 3$ , otherwise  $CH(R_v)$  can be obtained by direct computation.) Let  $l_v$  be any line passing by  $v$ , let  $T_v$  be any plane containing  $l_v$ , and let  $H_v$  be either halfplane of  $T_v$  defined by  $l_v$ ; selection of  $H_v$  determines a system of polar angles around  $l_v$ . Let  $(r_1, r_2, \dots, r_m)$  denote the cyclic list of rays in  $R_v$  (ordered by polar angle around  $l_v$ ); this ordered list can be constructed by sorting the  $m$  rays in  $R_v$  by angular order in  $O(\log m)$  time using  $O(m)$  processors on an EREW PRAM [Col88]. We partition this list into  $\sqrt{m}$  sublists,  $R_1, R_2, \dots, R_{\sqrt{m}}$ , each of size  $\sqrt{m}$ , so that  $R_1$  follows  $R_{\sqrt{m}}$  in the chosen order. We recursively construct the polygons  $\{\pi(R_j) | j = 1, \dots, \sqrt{m}\}$ , where  $\pi(R_j) = CH(R_j) \cap \Sigma$ . Note that if, for any  $1 \leq j \leq \sqrt{m}$ , the point set  $R_j \cap \Sigma$  is not contained in any hemisphere of  $\Sigma$ , then  $CH(R_j) = \mathfrak{R}^3$ , and thus  $CH(R_v) = \mathfrak{R}^3$ , i.e.,  $v$  is internal and the computation can stop; this condition will be detected by the recursive application of the merging process as described below.

At the last stage of the construction, we perform pairwise merges between all pairs of polygons in the set  $\{\pi(R_j) | j = 1, \dots, \sqrt{m}\}$ ; there are  $\binom{\sqrt{m}}{2} = O(m)$  such pairwise merges. In general, each

pairwise merge is accomplished by constructing two common supporting great circles for the pair of polygons, where each great circle is centered at  $v$  and passes by a point on the boundary of each polygon; we note here that it is easily shown that the task of computing the supporting great circles is nearly as simple as computing the corresponding common supporting lines in the plane case. Note that in the plane case, two common supporting lines are required for each pairwise merge of subhulls, and the convex hull of two subhulls is always a bounded convex set. This situation is somewhat altered in the present setting since if the two subhulls are not contained in a single hemisphere of  $\Sigma$ , then the convex span of their corresponding rays is  $\mathfrak{R}^3$ . Otherwise, as in the plane case, two supporting arcs (and two great circles) are needed to complete the merge. The sequential  $O(\log \sqrt{m})$  time technique of Dobkin and Kirkpatrick [DK90] for computing the common supporting lines to two convex polygons (each of size  $O(\sqrt{m})$ ) can easily be adopted to perform the pairwise merges. A useful feature of this technique (and the reason for adopting it here) as generalized to spherical polygons, is that it can easily distinguish between the above cases; initially it is assumed that two supporting great circles are required, but if at any point in the computation a supporting great circle cannot be computed, then the two subhulls are not contained in a single hemisphere, i.e.,  $v$  is internal. Thus, all pairwise merges can be accomplished in  $O(\log \sqrt{m})$  time using  $O(m)$  processors on a CREW PRAM; a CREW PRAM is needed because each of the  $O(\sqrt{m})$  subhulls is involved simultaneously in  $O(\sqrt{m})$  pairwise merges, i.e.,  $O(\sqrt{m})$  concurrent accesses.

After the pairwise merges are done, all tangents incident on a given point  $u(r)$  on  $\Sigma$  are sorted by slope, where the slope is measured with respect to a standard reference on a plane tangent to  $\Sigma$  at  $u(r)$  (such as a parallel-meridian system); if no two consecutive tangents form an angle  $> \pi$ , then  $u(r)$  corresponds to a ray  $r$  that is internal. If all rays are found to be internal, then  $v$  is also internal. If  $v$  is external, then, as in the planar algorithm, a standard list ranking operation constructs the neighborhood of  $v$ . The sorting [Col88] and list ranking [KR90] operations of this step can be implemented on an EREW PRAM in  $O(\log m)$  time using  $O(m)$  processors for each point of  $S$ . From the preceding discussion, we conclude that the final merge step is implementable in  $O(\log m)$  time with  $O(m)$  CREW PRAM processors.

The optimal  $O(\log n)$  time EREW PRAM planar convex hull algorithm of Miller and Stout [MS88] is similar in flavor to the above sketched algorithm, but in order to avoid the concurrent accesses the point set is initially partitioned into  $n^{1/4}$  groups each of size  $n^{3/4}$  and the pairwise merges are accomplished in  $O(1)$  stages with a simple sampling process. It is a simple matter to verify that this algorithm can also be adapted to compute  $\pi(R_v)$ , and thus  $CH(R_v)$ , to yield an EREW PRAM algorithm with same time and processor bounds.

Therefore, the time complexity of CLASSIFY( $v, S$ ) is  $O(\log m)$  using  $O(m)$  processors, where  $|S| = m$ , on either an EREW or CREW PRAM. Thus, we have the following theorem.

**Theorem 2.1** *The convex hull of  $n$  points in  $\mathfrak{R}^3$  can be found in  $O(\log n)$  time using  $O(n^2)$  processors on an EREW or CREW PRAM.*

**Proof:** For a given point  $v \in S$ , the algorithm CLASSIFY( $v, S$ ) will determine whether or not  $v$  is external to  $CH(S)$ , and moreover, if  $v$  is external, will compute  $v$ 's neighborhood on  $CH(S)$ . CLASSIFY( $v, S$ ) can be implemented in  $O(\log n)$  time using  $O(n)$  processors on a CREW PRAM (adapted from the optimal  $O(\log n)$  time CREW PRAM planar convex hull algorithm due to Atallah and Goodrich [AG86] and Aggarwal *et al.* [ACG<sup>+</sup>88]). Therefore, performing CLASSIFY for each  $v \in S$  will require  $O(\log n)$  and  $O(n^2)$  processors on a CREW PRAM.

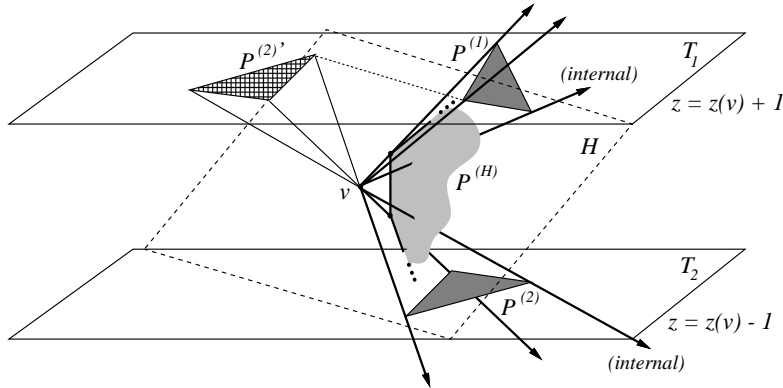


Figure 3: An alternative technique for computing  $CH(R_v)$ .

If an EREW PRAM algorithm is desired, then first  $n$  copies of the point set  $S$  are made, and then each point is classified using one of the available copies of  $S$  and the EREW PRAM version of CLASSIFY (adapted from the optimal  $O(\log n)$  time EREW PRAM planar convex hull algorithm of Miller and Stout [MS88]). ■

We end this section with a brief sketch of an alternative method for determining whether a particular point  $v$  is internal to  $CH(S)$ ; although the previous technique is perhaps intuitively more appealing, this alternative method does not involve spherical polygons but instead relies only on the construction of certain planar convex hulls and some additional simple computations. We consider the set  $R_v = \{r(v, p) \mid p \in S - \{v\}\}$  and the two planes  $T_1$  and  $T_2$ , whose respective equations are  $z = z(v) + 1$  and  $z = z(v) - 1$  (refer to Fig. 3). We let  $R_v^{(j)} = \{r \mid r \in R_v \text{ and } r \cap T_j \neq \emptyset\}$ ,  $j = 1, 2$ . Using the algorithm of [MS88] we construct, in time  $O(\log n)$  with  $O(n)$  processors on an EREW PRAM, the convex polygons  $P^{(j)} = CH(R_v^{(j)} \cap T_j)$ ,  $j = 1, 2$ . It is readily shown that  $v$  is external if and only if  $P^{(1)} \cap P^{(2)'} = \emptyset$ , where  $P^{(2)'}$  is the projection, through  $v$ , of  $P^{(2)}$  on  $T_1$ . This can be determined in time  $O(\log n)$  with a single processor [DK90]. If  $v$  is external, then  $CH(R_v)$  can be constructed as follows. In time  $O(1)$  with a single processor we determine a plane  $H$  intersecting all members of both  $R_v^{(1)}$  and  $R_v^{(2)}$ . ( $H$  is parallel to the plane determined by  $v$  and the two points of support of a segment separating  $P^{(1)}$  and  $P^{(2)'}$ , which was computed when testing if  $P^{(1)} \cap P^{(2)'} = \emptyset$ .) The central projections (from  $v$ ) of  $P^{(1)}$  and  $P^{(2)}$  on  $H$  are also obtained in  $O(1)$  time using  $O(n)$  processors, and can be merged into a single polygon  $P^{(H)}$  - the desired convex hull - in  $O(\log n)$  time with one processor. This alternative technique also yields  $CH(R_v)$  in time  $O(\log n)$  with  $O(n)$  processors on an EREW PRAM. Thus,  $CH(S)$  can be computed in  $O(\log n)$  time using  $O(n^2)$  processors on an EREW PRAM by this technique as well: first,  $n$  copies of the point set  $S$  are made, and then each point is classified using one of the available copies of  $S$ .

### 3 A More Efficient Algorithm

In this section we present our algorithm for constructing the convex hull of  $n$  points in  $\mathbb{R}^3$  in time  $O(\frac{1}{\alpha} \log n)$  using  $O(n^{1+\alpha})$  processors on a CREW PRAM, for any constant  $0 < \alpha \leq 1$ .

This algorithm utilizes the CLASSIFY procedure developed in the previous section; recall that we performed one CLASSIFY operation for each  $v \in S$ , and that the entire point set was used in each such invocation. Although the more efficient algorithm presented below may still perform a CLASSIFY operation for each  $v \in S$ , the work done by all the CLASSIFY operations in the aggregate will be less, i.e., while  $O(n)$  points may still be required in a few CLASSIFY operations, in total only  $O(n^{1+\alpha})$  points will be used, rather than the  $O(n^2)$  that were used in the previous algorithm. Thus, the trick to reducing the complexity of our initial algorithm is to select a subset of the original points that is sufficient to allow CLASSIFY to determine whether a particular vertex  $v$  is internal or external to  $CH(S)$ . An overview of the algorithm is given in Algorithm 3.1.

---

**Algorithm 3.1 3D-CONVEX-HULL( $S, n, \alpha$ )**

**Step 1.** Partition  $S$ , by  $z$ -coordinate, into  $n^\alpha$  groups, each of size  $n^{1-\alpha}$ ; denote the  $i$ th group by  $S_i$ .

**Step 2.** Recursively compute  $CH(S_i)$ ,  $\forall 1 \leq i \leq n^\alpha$ .

**Step 3.** Merge  $CH(S_i)$ ,  $1 \leq i \leq n^\alpha$ , to form  $CH(S) = CH(\cup_{1 \leq i \leq n^\alpha} CH(S_i))$ .

---

It is not difficult to analyze the complexity of Algorithm 3.1. Recall that we can sort all points in  $S$  in  $O(\log n)$  time with  $O(n)$  processors on an EREW PRAM using Cole's parallel merge sort [Col88]. Thus, in addition to the time for the initial sorting step, the time complexity of the above algorithm will satisfy the recurrence:

$$T(n) = T(n^{1-\alpha}) + M(n^\alpha, n^{1-\alpha}) \quad (1)$$

where  $M(n^\alpha, n^{1-\alpha})$  is the time required to merge  $n^\alpha$  linearly separated convex hulls, each of size  $n^{1-\alpha}$ . In the remainder of this section we will show that  $M(n^\alpha, n^{1-\alpha}) = O(\log n^{1+\alpha})$ , using  $O(n^{1+\alpha})$  processors on a CREW PRAM. It is easy to verify that  $O(n^{1+\alpha})$  processors are sufficient to perform the recursive computations of Step 2. This establishes that that  $T(n) = O(\frac{1}{\alpha} \log n)$ , using  $O(n^{1+\alpha})$  processors on a CREW PRAM.

We begin with some useful definitions. Consider two separable convex hulls  $CH(P)$  and  $CH(Q)$  and the convex hull of their union  $CH(P \cup Q)$ . As before, a vertex  $v \in CH(P)$  is *external* if it is a vertex of  $CH(P \cup Q)$ , and otherwise it is *internal*, and if  $v$  is external, the set of vertices that are incident to  $v$  on  $CH(P \cup Q)$  is  $v$ 's *neighborhood* and will be denoted by the sequence  $n(v)$ . In addition, a vertex  $v \in CH(P)$  is said to be a *seam* vertex if it is an external vertex and it is incident to some vertex  $w \in CH(Q)$  on  $CH(P \cup Q)$ , i.e., one of its neighbors on  $CH(P \cup Q)$  is a vertex of  $CH(Q)$ . External vertices that are not seam vertices will be referred to as *e-external* vertices, and external vertices that are seam vertices will be referred to as *s-external* vertices.

We let  $C_{ij}$  denote  $CH(S_i \cup S_j)$ ,  $i < j$ , and  $C = \{C_{ij} | 1 \leq i < j \leq n^\alpha\}$ , and make the following straightforward observations:

**Fact 3.1** If a vertex  $v$  of  $CH(S_i)$  or  $CH(S_j)$  is internal to  $C_{ij}$ , for some  $1 \leq i < j \leq n^\alpha$ , then it is also internal to  $CH(S)$ .

**Fact 3.2** If a vertex  $v \in CH(S_i)$  is e-external to  $C_{ij}$  and  $C_{j'i}$ ,  $\forall j, j'$  satisfying  $1 \leq j' < i < j \leq n^\alpha$ , then  $v$  is also external to  $CH(S)$ ; moreover, the vertices incident to such a vertex  $v$  on  $CH(S)$  will be the vertices that are incident to  $v$  on  $CH(S_i)$ , i.e.,  $n(v) = n_i(v)$ , where  $n_i(v)$  denotes  $v$ 's neighborhood on  $CH(S_i)$ .

**Fact 3.3** If a vertex  $v \in CH(S_i)$  is s-external to at least one  $C_{ij}$  or  $C_{j'i}$ , and is not internal to any  $C_{ij}$  or  $C_{j'i}$ ,  $\forall 1 \leq j' < i < j \leq n^\alpha$ , then  $v$  could be internal or external to  $CH(S)$ .

Therefore, if we compute the set  $C$  (i.e., complete the pairwise merges of all subhulls in the set  $\{CH(S_i) | 1 \leq i \leq n^\alpha\}$ ), then we need examine only those vertices that resulted external in *all* pairwise merges, and s-external in *at least one* pairwise merge. With this background, we now describe the merging process in Algorithm 3.2.

---

**Algorithm 3.2** MERGE( $CH(S_1), CH(S_2), \dots, CH(S_{n^\alpha})$ )

**Step 1.** Compute  $C_{ij}$ ,  $\forall 1 \leq i < j \leq n^\alpha$ . [There are  $\binom{n^\alpha}{2} = O(n^{2\alpha})$  such pairwise merges.]

**Step 2.** Let  $S^*$  denote the set of all vertices that resulted external in all pairwise merges in Step 1, and s-external in at least one pairwise merge. For each  $v \in S^*$ , construct the set  $A_v$ , which consists of all vertices that were s-external vertices adjacent to  $v$  in some pairwise merge, i.e., if  $v \in CH(S_i)$ ,  $w \in CH(S_j)$ ,  $i < j \leq n^\alpha$ , and  $w \in A_v$ , then  $w$  belongs to  $v$ 's neighborhood on  $C_{ij}$  (if  $1 \leq j < i$ , then  $w$  belongs to  $v$ 's neighborhood on  $C_{ji}$ ).

**Step 3.** Consider vertex  $v \in S^*$ ; assume that  $v \in CH(S_i)$  and let  $n_i(v)$  denote  $v$ 's neighborhood on  $CH(S_i)$ . For each such vertex  $v$ , determine if  $v$  is external or internal to  $CH(S)$  by performing the operation CLASSIFY( $v, A_v \cup n_i(v)$ ).

**Step 4.** Adjoin to the set  $\{v \mid v \in S^* \text{ and } v \text{ has been classified external in Step 3}\}$  the set of points that resulted e-external in all pairwise merges; these are the vertices of  $CH(S)$ . Form a doubly-connected-edge-list (DCEL) representation of  $CH(S)$ .

---

Next, we establish the correctness of MERGE. We have already argued that only the vertices in  $S^* \subseteq S$  need be considered; we now argue that Step 3 of MERGE correctly classifies each  $v \in S^*$ . Indeed, the following lemma establishes that the point set  $S$  is contained in  $CH(R_v^*)$ , where  $v \in S^* \cap CH(S_i)$  and  $R_v^* = \{r(v, p) \mid p \in A_v \cup n_i(v)\}$ . It then follows that  $CH(S) \subset CH(R_v^*)$ , which implies that (i)  $v$  is external to  $CH(S)$  if and only if it is external to  $CH(R_v^*)$ , (or equivalently,  $v$  is internal to  $CH(S)$  if and only if it is internal to  $CH(R_v^*)$ ), and (ii) if  $v$  is external, then  $v$ 's neighborhood on  $CH(S)$  must lie on the boundary rays of  $CH(R_v^*)$ , i.e.,  $CH(R_v^*)$  identifies the sequence  $n(v)$ .

**Lemma 3.1** *If  $v \in S^* \cap CH(S_i)$ , then  $S \subset CH(R_v^*)$ , where  $R_v^* = \{r(v, p) \mid p \in A_v \cup n_i(v)\}$ .*

**Proof:** Consider some  $v \in S^* \cap CH(S_i)$ , and let  $A_v^* = A_v \cup n_i(v)$ . To obtain a contradiction, assume there is some  $w \in S$  such that  $w \notin CH(R_v^*)$ . It is easy to see that  $w \notin S_i$ ; indeed, since  $n_i(v) \subset A_v^*$ , it must be that  $CH(S_i) \subset CH(R_v^*)$ , i.e.,  $CH(S_i) \subset CH(R_i) \subset CH(R_v^*)$ , where  $R_i = \{r(v, p) \mid p \in n_i(v)\}$ . Thus, it must be that  $w \in S_j$ , for some  $j \neq i$ ; assume without loss of

generality that  $i < j$ . Clearly  $v$  is external to  $C_{ij}$ , because otherwise  $v$  would not be a vertex of  $S^*$ . Let  $n_{ij}$  denote  $v$ 's neighborhood on  $C_{ij}$ , and let  $R_{ij}$  denote the set of rays  $\{r(v, p) \mid p \in n_{ij}\}$ . Note that  $CH(S_j) \subset C_{ij} \subset CH(R_{ij})$ , i.e.,  $w \in CH(R_{ij})$ . However, this implies that  $w \in CH(R_v^*)$ , since by definition of  $A_v^*$  we have  $n_{ij} \subseteq A_v^*$ , which contradicts our assumption that  $w \notin CH(R_v^*)$ . ■

Lemma 3.1 establishes the correctness of MERGE, and thus, we now turn our attention to its complexity. We recall that the  $O(\log^2 n)$  time,  $O(n)$  processor CREW PRAM algorithm of [AP92] for computing the convex hull of an arbitrary point set in  $\mathbb{R}^3$ , uses a bisect-and-conquer strategy in which, at each of  $O(\log n)$  stages, two separable convex hulls,  $CH(P)$  and  $CH(Q)$ , are merged in  $O(\log n)$  time to form  $CH(S) = CH(P) \cup CH(Q)$ , where  $|CH(P)| = |CH(Q)| = n$ . In order to achieve this time bound, the subhulls  $CH(P)$  and  $CH(Q)$  (represented by DCELs) are preprocessed so that they can be tested for intersection with arbitrary lines and planes quickly. In particular, to achieve the time and processor bounds required here, each intersection test must be accomplished in  $O(\log n)$  time using  $O(n^\alpha)$  processors; in addition, the preprocessing of the subhulls to construct structures suitable for these intersection queries should be accomplished in  $O(\log n)$  time using  $O(n)$  processors. There are a couple data structures which meet these requirements: the three-dimensional extension of the *bridged separator tree* [LP77, EGS86] introduced by Tamassia and Vitter [TV91, TV90], and the *hierarchical representation* [DK90] as optimized by Cole and Zajicek [CZ90].

The technique used to accomplish the merging process in [AP92] determines, for each edge  $e \in CH(P) \cup CH(Q)$ , whether or not  $e$  is an internal, external, or seam edge of  $CH(CH(P) \cup CH(Q))$ ; these classifications can clearly be used to determine whether or not a vertex is internal, e-external or s-external to the merged hull. Thus, the merging process of [AP92] can be used to implement Step 1 of MERGE; since there are  $O(n^{2\alpha})$  pairwise merges of convex hulls each of size  $O(n^{1-\alpha})$ , Step 1 requires  $O(\log n^{1-\alpha})$  time using  $O(n^{2\alpha} \cdot n^{1-\alpha}) = O(n^{1+\alpha})$  processors.

Since a polytope resulting from each merge in Step 1 can have size  $O(n^{1-\alpha})$ , and there are  $O(n^{2\alpha})$  such merges, the number of vertices in the set  $A = \cup_{v \in S^*} A_v$  can be as large as  $O(n^{1+\alpha})$ , where the sets  $A_v$  are as defined in Step 2; the sets  $A_v$  can be formed by a simple sorting process in time  $O(\log n^{1+\alpha})$  using  $O(n^{1+\alpha})$  processors [Col88]. Although a single set  $A_v$  can be of size  $O(n)$ , the CLASSIFY operations of Step 3 will require  $O(\log n)$  time and  $O(n^{1+\alpha})$  processors in the aggregate because  $|A| = O(n^{1+\alpha})$ . Since the neighborhood of any vertex external to  $CH(S)$  was identified in Step 3, it is a trivial matter to verify that in Step 4 a doubly-connected-edge-list (DCEL) representation of  $CH(S)$  can be constructed in constant time using  $O(n)$  processors.

Thus, the total complexity of the merging process is  $O(\log n^{1+\alpha})$  time using  $O(n^{1+\alpha})$  processors on a CREW PRAM, i.e.,  $M(n^\alpha, n^{1-\alpha}) = O(\log n^{1+\alpha})$  using  $O(n^{1+\alpha})$  processors. Plugging this value into Equation (1) yields  $T(n) = T(n^{1-\alpha}) + O(\log n^{1+\alpha}) = O(\frac{1}{\alpha} \log n)$  using  $O(n^{1+\alpha})$  processors on a CREW PRAM, which establishes the following theorem. We note that although we can afford to make  $n^\alpha$  copies of the point set, we cannot implement this algorithm on an EREW PRAM because the algorithm [AP92] used for the pairwise merges in Step 1 requires concurrent reads.

**Theorem 3.1** *The convex hull of a set of  $n$  points in three-dimensional space can be computed in  $O(\frac{1}{\alpha} \log n)$  time using  $O(n^{1+\alpha})$  processors on a CREW PRAM, for any constant  $0 < \alpha \leq 1$ .*

## 4 Acknowledgement

We are grateful to Guenter Rote for his helpful comments which greatly improved the presentation of the CLASSIFY algorithm. We would also like to thank Jiri Matoušek for his insightful comments, and an anonymous referee for pointing out that the  $O(\log n)$  time and  $O(n^2)$  processor algorithm of Section 2 can be implemented on an EREW PRAM.

## 5 Conclusion

We have described an  $O(\frac{1}{\alpha} \log n)$  time algorithm for computing the convex hull of a three-dimensional point set, which uses  $O(n^{1+\alpha})$  processors on a CREW PRAM, for any constant  $0 < \alpha \leq 1$ . All other deterministic parallel algorithms for this problem (i.e., [Cho80, ACG<sup>+</sup>88, DK89, AP92, Goo93]) have time bounds of at least  $O(\log^2 n)$ . In addition, this is the first deterministic parallel algorithm for the three-dimensional convex hull problem that does not employ the bisect-and-conquer paradigm of the optimal sequential algorithm of Preparata and Hong [PH77]; a recent exception is the work-optimal,  $O(\log^2 n)$  time, algorithm due to Goodrich [Goo93], which is obtained by derandomizing the time- and work-optimal randomized algorithm of Reif and Sen [RS92]. Our algorithm uses a many-way divide-and-conquer strategy to achieve optimal  $O(\log n)$  time on an exclusive write PRAM. Such an objective appears difficult to obtain using the bisect-and-conquer paradigm since it requires  $O(\log n)$  merging phases. However, although time-optimal, the described algorithm is not work-optimal.

## References

- [ACG<sup>+</sup>88] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, and C. Yap. Parallel computational geometry. *Algorithmica*, 3:293–327, 1988.
- [AG86] M. J. Atallah and M. T. Goodrich. Efficient parallel solutions to some geometric problems. *J. Parallel Distrib. Comput.*, 3:492–507, 1986.
- [AG88] M. J. Atallah and M. T. Goodrich. Parallel algorithms for some functions of two convex polygons. *Algorithmica*, 3:535–548, 1988.
- [And79] A. M. Andrew. Another efficient algorithm for convex hulls in two dimensions. *Inform. Process. Lett.*, 9(5):216–219, 1979.
- [AP92] N. M. Amato and F. P. Preparata. The parallel 3D convex hull problem revisited. *Internat. J. Comput. Geom. Appl.*, 2(2):163–173, 1992.
- [AP93] N. M. Amato and F. P. Preparata. An  $NC^1$  parallel 3D convex hull algorithm. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 289–297, 1993.
- [CDR86] S. A. Cook, C. Dwork, and R. Reischuk. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM J. Comput.*, 15:87–97, 1986.
- [Che90] D. Z. Chen. Efficient geometric algorithms in the EREW-PRAM. In *Proc. 28th Allerton Conf. Commun. Control Comput.*, pages 818–827, 1990.

- [Cho80] A. L. Chow. *Parallel algorithms for geometric problems*. Ph.D. thesis, Dept. Comput. Sci., Univ. Illinois, Urbana, IL, 1980.
- [CM92] B. Chazelle and J. Matoušek. Derandomizing an output-sensitive convex hull algorithm in three dimensions. Technical report, Dept. Comput. Sci., Princeton Univ., 1992.
- [Col88] R. Cole. Parallel merge sort. *SIAM J. Comput.*, 17(4):770–785, 1988.
- [CS89] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.
- [CZ90] R. Cole and O. Zajicek. An optimal parallel algorithm for building a data structure for planar point location. *J. Parallel Distrib. Comput.*, 8:280–285, 1990.
- [DK89] N. Dadoun and D. G. Kirkpatrick. Parallel construction of subdivision hierarchies. *J. Comput. Syst. Sci.*, 39:153–165, 1989.
- [DK90] D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra – a unified approach. In *Proc. 17th Internat. Colloq. Automata Lang. Program.*, volume 443 of *Lecture Notes in Computer Science*, pages 400–413. Springer-Verlag, 1990.
- [EGS86] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15:317–340, 1986.
- [ES91] H. Edelsbrunner and W. Shi. An  $O(n \log^2 h)$  time algorithm for the three-dimensional convex hull problem. *SIAM J. Comput.*, 20:259–277, 1991.
- [GG91] M. Ghouse and M. T. Goodrich. In-place techniques for parallel convex hull algorithms. In *Proc. 3rd ACM Sympos. Parallel Algorithms Architect.*, pages 192–203, 1991.
- [Goo93] M. T. Goodrich. Geometric partitioning made easier, even in parallel. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 73–82, 1993.
- [Gra72] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inform. Process. Lett.*, 1:132–133, 1972.
- [JáJ92] J. JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley, Reading, MA, 1992.
- [KR90] R. M. Karp and V. Ramachandran. Parallel algorithms for shared memory machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 869–941. Elsevier/The MIT Press, Amsterdam, 1990.
- [KS86] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 15:287–299, 1986.
- [LP77] D. T. Lee and F. P. Preparata. Location of a point in a planar subdivision and its applications. *SIAM J. Comput.*, 6:594–606, 1977.
- [MS88] R. Miller and Q. F. Stout. Efficient parallel convex hull algorithms. *IEEE Trans. Comput.*, C-37(12):1605–1618, 1988.

- [PDW92] W. Preilowski, E. Dahlhaus, and G. Wechsung. New parallel algorithms for convex hull and triangulation in 3-dimensional space. In *Proc. Internat. Sympos. Math. Found. Comput. Sci. (MFCS 1992)*, volume 629 of *Lecture Notes in Computer Science*, pages 442–450. Springer-Verlag, 1992.
- [PH77] F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, 20:87–93, 1977.
- [PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction*. Springer-Verlag, New York, NY, 1985.
- [Rei93] J. H. Reif, editor. *Synthesis of Parallel Algorithms*. Morgan Kaufmann, San Mateo, CA, 1993.
- [RS92] J. H. Reif and S. Sen. Optimal parallel randomized algorithms for three-dimensional convex hulls and related problems. *SIAM J. Comput.*, 21(3):466–485, June 1992.
- [TV90] R. Tamassia and J. S. Vitter. Optimal cooperative search in fractional cascaded data structures. In *Proc. 2nd ACM Sympos. Parallel Algorithms Architect.*, pages 307–316, 1990. To appear in *Algorithmica*.
- [TV91] R. Tamassia and J. S. Vitter. Parallel transitive closure and point location in planar structures. *SIAM J. Comput.*, 20(4):708–725, 1991.